N88-16363

TEMPORAL AND CONTEXTUAL KNOWLEDGE IN MODEL-BASED EXPERT SYSTEMS: Ford Aerospace's Paragon Project

Tihamer Toth-Fejel and Dennis Heher
Ford Aerospace and Communications Corporation
Sunnyvale, California 94089-1198

ABSTRACT: This paper introduces Paragon, a general-purpose environment for building model-based expert systems. The focus is on contextual and temporal representation, with time considered a type of context.

Conceptually, a Paragon knowledge base is a highly constrained semantic network. Its interfaces enable the domain expert to make knowledge explicit, prevent the proliferation (and potential contradiction) of preconditions found in rule-based systems, and make the knowledge base highly partitionable for parallel processing. Paragon automatically generates LISP code from the knowledge base. Executing this code provides a simulation that allows the domain expert to observe the explicitly described behavior and verify the validity of the knowledge base.

Paragon has been demonstrated in domains that are understood well enough to be modeled, such as satellite diagnostics, ground station diagnostics, and satcom network monitoring.

INTRODUCTION: Temporal and contextual representation in expert systems is a difficult area in Artificial Intelligence (AI). Most expert systems consider time as a special type of context, as does Paragon. In rule-based systems, context is represented by the premise of each rule. Since the context in which a rule takes effect is globally referenced, the premises of rules become longer as the knowledge base gets larger. Partitioning rules into contextually similar sets only delays the inevitable.

At Ford Aerospace, when preliminary calculations showed that the domain of satellite diagnostics would require more than 100,000 rules, the researchers sought another technique [4][7]. The model-based semantic network approach, exemplified in the Calisto project [6], seemed promising for making large problems tractable.

GOALS: The satellite diagnostics domain is well understood; however, domain experts are still expensive and difficult to obtain. Therefore, it is desirable to train them in a minimal amount of time (one week) and immediately place them in front of a workstation running Paragon. Then the domain expert should describe a satellite's components, the causal and compositional relationships between them, and their behavior in terms of states, transitions, and events. The conceptual representation described should exhibit cognitive resonance. In other words, not only should Paragon be user-friendly, but it should also represent the high-level concepts and relations that the domain expert uses when thinking. Finally, Paragon should manipulate these concepts and relations in the same way as the expert does, thus accomplishing the intended tasks of fault diagnosis, analysis, correction, and planning in well-understood domains. As in most AI problem areas, everything hinges on knowledge representation.

PARAGON REPRESENTATION: The Paragon representation can be defined at five layers [2], as outlined by Ferguson [3].

At the **application layer**, Paragon has been demonstrated in many domains: satellite diagnostics, satellite network monitoring, and ground station diagnostics, with expected demonstration of planning and pattern recognition in the satellite domain. At this layer, Paragon enables the domain expert to specify (ie. create, name, and link) domain-specific knowledge about batteries, heaters, their thermal and electrical relationships, commands, procedures, and other physical and non-physical ideas. The domain expert never sees LISP code, and touches the keyboard only when assigning names.

At the **conceptual layer**, Paragon provides primitives such as Concepts (semantic net nodes), and Relations (semantic net links). Central to the issue of knowledge acquisition, Paragon makes it possible for the domain expert to create user-defined, but Paragon-constrained, Concepts and Relations.

At the **epistemological layer**, Paragon provides six types of concepts that can be named and linked to other concepts (Figure 1 illustrates the examples):

1) Primitive Concepts (Primitives) represent instantiations of physical and nonphysical objects in the real world (ie. Battery1, Switch2). They gather Attribute Concepts together under a single name, and are the primary focus of behavior, causal relations, and compositional relations.

2) Definition Concepts (Classes) are generic covering sets of other Primitives or Classes, and allow the grouping of concepts by classification (ie. Battery, Heater).

3) Attribute Concepts (Attributes) describe properties of Primitives and Classes (ie. Voltage) and contain the name, type, and value of those properties.

4) Composite Concepts (Composites) are compositional groupings of concepts (ie. Composite Heater1 "has parts" HeatingElement and Switch2).

5) Event Concepts (Events) describe changes to Attribute values of a Primitive while in a particular state, and are equational in nature (for example, the Voltage of Battery1 is divided by the Resistance of Heater1 => the Current of Battery1.

6) State Concepts (States) collect the Events that occur while a Primitive is in a particular state (ie. Charge or Discharge).

Relations (the instances of which are also called links) contain a minimal amount of information and can be of four main types:

1) Specialization (or Classification) Relations describe the relations between classes and their subclasses or members. For example, there exists a Specialization Relation between Battery and Battery1. Inheritance of States, Events, and Attributes can occur in both directions along this link, saving considerable time in knowledge acquisition.

2) Compositional Relations describe the relations between concepts and their composites and/or components. One example was given in describing Composite Concepts.

3) Causal Relations describe how Primitives affect each other. For example, Battery1 POWERS HeatingElement1. Every Causal Relation is defined by the domain expert.

4) Transition Relations (Transitions) describe conditions under which "control" of a Primitive can switch between States, and contains a LISP expression which evaluates to true or false.

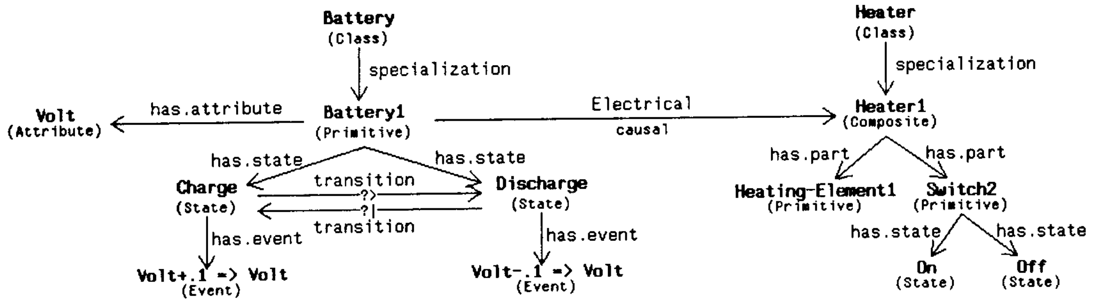

Figure 1. Paragon Knowledge Base

One of the most important aspects of Paragon with respect to contextual representation is that it enforces the principle of locality. This principle constrains access to information by requiring an explicit relation to be defined between concepts [3]. An Event can affect only one local Attribute, and while it can access all the other Attributes of the Event's Primitive, it can only access external attributes through explicit causal relations that are associated with the Event's Primitive. Transitions are constrained by the same principle.

The **logical layer** can be interpreted as describing how Paragon handles the ANDing and ORing of multiple relations and concepts. For example, if two concepts are electrically connected by a number of digital electrical lines, then the domain expert can model it either with a single electrical relation, or many.

At the **implementation layer**, LISP hash arrays were used to represent both concepts and relations, while all Paragon interface and core functionality was implemented in LISP. For reasons of practicality, additional liberties were taken to compromise the "pure" Paragon representation with frame-based constructs.

After the domain expert finishes describing each piece of the domain, Paragon generates a simulation model in LISP source code. This code can then be executed, and its behavior observed via graphic active images and values. The model can be tested as if it was a piece of physical hardware, and if it demonstrates the same input/output functionality as a good device, then the domain expert knows that the knowledge base is correct [8]. This is a bold but unsupported claim. The developers of Paragon have not been able to mathematically prove truth preservation or logical correctness. However, humans seem to do intellectual tasks quite well without such rigorous proofs, and even computer programs have been quite useful without strict logical foundations. In the experience of Paragon users, a mistake in the description can appear either as wildly oscillating behavior or as no behavior at all. In other cases, the description error and resulting mis-modeling is more subtle, just as in physical hardware.

Faults are not modeled in Paragon, because such modeling would be self-defeating. This is because a device can fail in many more ways than it can work properly. The lack of fault models has not proven to be a handicap, because Paragon reasoning modules can pinpoint faults to whatever level the domain has been modeled. Current research on causal analysis and planning indicate that the Paragon representation is ideal for determining the cause of failures, and possibly even for finding recovery procedures, though some very high level of fault representation may turn out to be useful.

CONTEXTUAL REPRESENTATION: In a Paragon Knowledge Base, every object of the target domain is represented by a Primitive. This Primitive may have many States, with different Events changing their corresponding Attribute values differently in each particular State. In the example illustrated in figure 1, Battery1 has States Discharge and Charge, each containing an Event which modifies the value of Volt. The Process Definition Interface allows the domain expert to specify States and Transitions between States. There are two Transitions connecting Discharge and Charge, illustrated by ?> (conditionally true) and ?| (conditionally false).

To keep the simulator from endlessly looping through states via these transitions, the domain expert must specify the conditions (or context) under which the transition will occur, assuming that the Primitive is currently in the "from" state of that particular transition link. In many ways, this condition is similar to the premise of a rule, and caution must be exercised to prevent the proliferation of precondition clauses (that define a context) and the "ad-hoc-ness" that results from too much flexibility.

The purpose of the Context Specification Interface (CONTXSPEC) is to address the above problem by enforcing the principle of locality. At the implementation level, CONTXSPEC outputs a condition -- a piece of LISP code that evaluates to either true or false. On the input side, CONTXSPEC must satisfy the same four criterion that any representation of reality must provide [6]:

1) Completeness - represent all relevant and necessary knowledge in the domain.
2) Precision - provide appropriate granularity of knowledge.
3) Clarity - lack ambiguity in interpretation.
4) Cognitive Resonance - use the same concepts the domain expert does.

CONTXSPEC satisfies the criterion of completeness by providing the domain expert with relational operators (=, <, >, etc) and local Primitive Attributes (Voltage of Battery1, Temperature of HeaterA). Only Attributes of the States' Primitive and those passed in by causal links can be accessed, in accordance with the principle of locality. This principle of locality limits communication between concepts by requiring all cause/effect relationships to be specified. While very frustrating at times, this strict limitation engenders a number of significant advantages:

1) When the domain expert wants to access a non-local Attribute, but CONTXSPEC doesn't provide that access, then he or she will realize that a causal relation has not yet been made explicit.

2) Prevents the proliferation of preconditions by tying the context to the location of the nodes and links within the semantic net.

3) Makes each transition self-contained with no side-effects, a property that makes the Paragon KB ideally suited for parallel processing.

4) Finally, the principle of locality restricts the impossibly large number of attributes CONTXSPEC must otherwise display to a manageable number.

By mousing four times, the domain expert can specify a simple condition such as "The Voltage of SolarArray1 is > 10", while CONTXSPEC prevents low-level errors such as mistyping or selecting the wrong menu. Most of the conditions in Transitions are this simple, though much more complicated ones are possible with logical operators AND, OR, and/or NOT. Most of the complexity ends up in the state graph, with its multiplicity of states and possible transitions. If the state diagram is too complicated, this is an indication that the concept should be broken down into its components, or that some states may be merged, unfortunately at a loss of modeling detail. Internally, this condition is represented by a case-grammar-like sentence, which is very easy to translate to LISP, establishing a condition in a Transition between two States. This condition can be true or false, depending on the overall state of the world, or in this example, the value of the Attribute Voltage of the Primitive SolarArray1. The evaluation of Events during the execution of simulation code causes Attributes' values to change.

Originally, Paragon gave the domain experts the capability to represent complex equations inside the Transition's conditions. Unfortunately, equations hide large amounts of implicit knowledge. Therefore Paragon was changed to prevent the domain experts from entering equations anywhere except in the Events. This increased difficulty in knowledge acquisition was caused by the necessity to break up complicated fomulas into representations more amenable to automated reasoning.

TEMPORAL REPRESENTATION: Reasoning about time is presently a very debated issue in AI. However, some ideas are generally agreed upon, for example Allen's relations on convex time intervals [1]. Early in Paragon's development, sets of algebraic mappings were found between any two intervals to produce a third interval (ie. Plus, Minus, Cross-Product, etc). With a set of relations and mappings, it was assumed that a useful algebra could be integrated into Paragon. The Temporal Specification Interface (TSI) was developed to quickly specify complicated temporal expressions. Unfortunately, a non-trivial algebraic group for time intervals was not found; fortunately, this lack of success didn't matter. During the development of TSI and the search for a temporal algebraic group, the domain experts used Paragon concepts to simulate clocks and timers whenever they needed them. Upon closer examination, it was found that Allen's relations could be implemented in Paragon at the Attribute-State-Event level, so TSI was never integrated into Paragon. In addition, Paragon has the ability to do multi-interval comparisons, which Ladkin showed to be infeasible when representing at the interval-relation level [5].

It turns out to be quite simple to set up very complicated timed, conditional or asynchronous cycles (or non-cyclic temporal state changes) in the state transition graph of a Primitive. For example, to simulate the sun-shade cycles that a solar array experiences in orbit, the Primitive SolarArray1 has an Attribute Clock, which is reset in States StartShade and EndShade by identical Events (0 => Clock) and set by (Clock+1 => Clock) in States Sun and Shade. The description of SolarArray1's behavior is completed by specifiying theTransitions to become true when Clock values are equal to 51 and 39 respectively.

As in most rule-based systems, temporal representation is a special case of context in Paragon. This is because at a certain level of granularity and ignorance, time (and context) can be considered a cause. In a rule, the premise can be thought of as causing the consequent. This paradigm may lead to the nonsensical belief that Monday causes Tuesday. Well, not exactly. The events that occur during Monday (like the passage of time) cause the Transition between Monday and Tuesday to become true, allowing the change of state. Given ignorance of the structure of the solar system and of the naming convention regarding 24-hour periods, it is perfectly acceptable for automated causal diagnosis to conclude that Monday causes Tuesday.

CONCLUSION: This paper presents a basic paradigm that allows representation of physical systems, with a focus on context and time. Paragon provides the capability to quickly capture an expert's knowledge and represent that knowledge in a cognitively resonant manner. From that description, Paragon creates a simulation model in LISP, which when executed, verifies that the domain expert did not make any mistakes. The Achilles heel of rule-based systems has been the lack of a systematic methodology for testing, and Paragon's developers are certain that the model-based approach overcomes that problem. The reason this testing is now possible is that software, which is very difficult to test, has in essence been transformed into hardware.

REFERENCES

[1] Allen, J., Maintaining Knowledge about Temporal Intervals, **Readings in Knowledge Representation**, edited by Brachman, R., and Levesque, H., Morgan Kaufmann, 1985, pp. 510-521.

[2] Brachman, R., On the epistemological Status of Semantic Networks, **Readings in Knowledge Representation**, edited by Brachman, R., and Levesque, H., Morgan Kaufmann, 1985, pp. 191-215.

[3] Ferguson, J. C., Beyond Rules: The Next Generation of Expert Systems, **Proceedings of the Air Force Workshop on AI Applications for Integrated Diagnostics**, May 1987.

[4] J. Ferguson, R. Siemens and R. Wagner, Starplan: A Satellite Anomaly Resolution and Planning System, from J. Kowalik, ed., **Coupling Symbolic and Numerical Computing in Expert Systems**, pp. 273-281.

[5] Ladkin, P., Time Representation: A Taxonomy of Interval Relations, **AAAI-86 Proceedings**, August 11-15, 1986, pp. 360-366.

[6] Sathi, A., Fox, M., and Greenberg, M., Representation of Activity Knowledge for Project Management, **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. PAMI-7, No. 5, September, pp. 531-552.

[7] Siemens,R., Golden, M., and Ferguson, J. C., "Starplan II: Evolution of an Expert System", **AAAI 86 Proceedings**, pp. 844-850.

[8] Toth-Fejel, T.T., **Self-Test: From Simple Circuits to Self-Replicating Automata**, Master's Thesis, University of Notre Dame, 1984, pp. 160-161.